# I. Sortare - generalități

Sortarea tablourilor unidimensionale (vectori) reprezintă rearanjarea elementelor acestora astfel încât între valorile lor există o relație de ordine. (crescător/descrescător, A-Z/Z-A, etc.).

1. Sortare prin selecție directă;
2. Sortare prin selecția minimului/maximului;
3. Sortare prin metoda bulelor (BubbleSort);
4. Sortare prin inserție;
5. Sortare prin determinarea poziției prin numărare;
6. Sortare rapidă (QuickSort).

Algoritmii următori ordonează vectorii crescător.

# II. Algoritmi de sortare a vectorilor

## 1. Selecție directă

|  |  |
| --- | --- |
| Această metodă presupune aducerea pe primul loc a celui mai mic element dintre cele n elemente ale tabloului pe prima poziție, apoi aducerea celui mai mic element dintre cele n-1 ramase pe cea de a doua poziție ș.a.  Această „aducere” se realizează prin comparări și interschimbări succesive a valorii curente cu cele ce o urmează în vector.  Complexitatea algoritmului este n2. | int aux;  for(int i=0;i<n-1;i++) for(int j=i+1;j<n;j++) if(a[j]<a[i])  {  aux=a[i];  a[i]=a[j];  a[j]=aux;  } |

## 2. Sortare prin selecția minimului/maximului;

|  |  |
| --- | --- |
| Această metodă presupune interschimbarea valorii curente cu valoarea cea mai mica dintre toate elementele vectorului începând cu cea curentă. **Exemplu**: elementul curent este primul din tablou. Pornind de la acesta se caută valoarea minimă din tablou și apoi se interschimbă cu cea curentă, după care algoritmul se reia plecând de la al doilea element. Procedeul continuă până la penultimul element.  Complexitatea algoritmului este n2. | int aux, Min, pozMin;  for(int i=0;i<n;i++)  {  Min=a[i], pozMin=i; for(int j=i+1;j<n;j++) if(a[j]<Min)  {  Min=a[j]; pozMin=j;  } aux=a[i]; a[i]=a[pozMin]; a[pozMin]=aux;  } |

## 3. Sortare prin metoda bulelor (BubbleSort);

|  |  |
| --- | --- |
| Această metodă presupune parcurgerea tabloului în mod repetat și compararea fiecărui element din tablou cu succesorul său. Dacă nu sunt în ordine cele două elemente se interschimbă.  Dacă la o parcurgere se face măcar o interschimbare, vectorul se parcurge din nou. Vectorul este sortat dacă **la o parcurgere nu se face nici o interschimbare**.  Complexitatea algoritmului în cazul general este n2. În cazul cel mai favorabil (vectorul este ordonat crescător) complexitatea este liniară (n). | int gata=0, aux; while(!gata)  { gata=1;  for(int i=0;i<n-1;i++) if(a[i]>a[i+1])  {  aux=a[i]; a[i]=a[i+1]; a[i+1]=aux; gata=0;  }  } |

## 4. Sortare prin inserție;

|  |  |
| --- | --- |
| Începând cu al doilea element al tabloului, fiecare element este inserat pe poziția adecvată în subtabloul care îl precedă.  Această metodă presupune inserarea elementului curent din vector pe o poziție astfel încât elementele să rămână ordonate pană la poziția curentă, astfel: elementul curent este extras din vector, apoi este identificată poziția pe care ar trebui așezat astfel încât ordinea până la elementul curent să fie păstrată, după care se face efectiv inserția.  Inserția elementelor se poate face și într-un al doilea vector, ale cărui elemente vor fi copiate la final în vectorul inițial.  Complexitatea algoritmului în cazul general este n2. În cazul cel mai favorabil (vectorul este ordonat crescător) complexitatea este liniară (n). | int aux, j;  for(int i=1;i<n;i++)  {  aux=a[i]; j=i-1;  while(j>=0 && a[j]>aux)  {  a[j+1]=a[j]; j--; }  a[j+1]=aux;  } |

## 5. Sortare prin determinarea poziției prin numărare;

|  |  |
| --- | --- |
| Această metodă presupune numărarea pentru fiecare element al tabloului, a numărului de valori mai mici decât acesta într-un vector de apariții (ap). Această informație este apoi utilizată pentru a determina poziția fiecărui element în tabloul ordonat. Daca sunt cinci elemente mai mici decât x în tablou, atunci în tabloul ordonat x se va afla pe poziția a șasea.  Pentru așezarea elementelor în ordinea dată de vectorul de apariții se utilizează un al treilea vector (dest).  La final se copiază elementele din vectorul auxiliar (dest) în cel inițial. | int i, j, dest[n], ap[n]; for(int i=0;i<n;i++) ap[i]=0; for(int i=0;i<n-1;i++) for(int j=i+1;j<n;j++) if(a[i]>a[j]) ap[i]++; else ap[j]++; for(int i=0;i<n;i++) dest[ap[i]]=a[i]; for(int i=0;i<n;i++) a[i]=dest[i]; |

## 6. Sortare rapidă (QuickSort)

|  |  |
| --- | --- |
| Sortarea rapidă este o metodă de sortare eficientă ce se bazează pe ideea alegerii unui element denumit pivot și așezării acestuia în cadrul vectorului astfel încât toate elementele mai mici decât acesta să se găsească în stânga sa iar cele mai mari în dreapta.  În acest mod pivotul se va afla pe poziția finală în cadrul vectorului ordonat.  Algoritmul se reia pentru elementele din stânga și din dreapta pivotului.  Alegerea pivotului influențează performanța algoritmului în mare măsură.  Complexitatea algoritmului în cazul general este n\*log2n iar în cazul cel mai defavorabil n2. | int a[100];  int pozitie(int st,int dr)  {  int aux,stare=0; while(st<dr)  {  if(a[st]>a[dr])  {  aux=a[st]; a[st]=a[dr]; a[dr]=aux; stare=!stare;  }  if(stare==0) dr--; else st++;  }  return st;  }  void qsort(int st,int dr)  { int p; if(st<dr)  {  p=pozitie(st,dr); qsort(st,p-1); qsort(p+1,dr);  }  } |